> **Draft note** - This is a live working draft adapted from talk notes at a conference session on 11 Mar 2026. More slides are still coming in, so this post is intentionally being built in stages.
>
> **TL;DR** Agents are no longer just copilots that answer questions. They already read documents, classify records, reconcile messy inputs, and act inside real business workflows. That makes the security problem much broader than prompt safety. A production agent has an **infrastructure surface**, an **authority surface**, and an **influence surface**. The core claim of this talk is sharp: we already have an execution plane for agents, but we still lack a mature **security control plane** for bounding autonomy, enforcing policy, preserving auditability, and keeping humans meaningfully in the loop.

---

1 Agents are already in production

The talk opens by rejecting the idea that agents are still a future-looking toy category.

The speaker gives concrete enterprise examples:

- **Bank of Singapore** uses an agent to read financial documents and write **KYC reports**, reportedly shrinking turnaround from **10 days to 1 hour**.
- **Salesforce** has reported **millions of customer conversations** handled autonomously, with a large share resolved without human intervention.
- A **Gartner** forecast suggests task-specific agents will appear in a large share of enterprise applications.

The exact percentages and adoption curves matter less than the framing move:

agent hardening is now a production problem, not a lab problem.

That matters because once agents touch:

- customer communication
- financial workflows
- internal documents
- toolchains
- sensitive data

then a failure is no longer “the model said something weird.” It becomes:

- a data leak
- a workflow manipulation
- a false business record
- a silent operational error
- or a costly compliance incident

---

2 The dangerous part is not the chat interface - it is the workflow

One of the strongest slides in the talk uses a deliberately ordinary example:

perform bookkeeping from email

At first glance, that sounds like a reasonable automation task. But once you expand it into actual steps, the surface area becomes obvious.

The agent is expected to:

- read email
- extract invoices and receipts
- extract bank statements
- handle duplication
- determine classifications
- infer reconciliation rules
- handle split bills and partial payments

That is not “AI summarization.” That is a multi-step operational workflow touching:

- untrusted inbound content
- highly sensitive financial data
- business logic
- reconciliation assumptions
- edge cases that can hide fraud or error

This is a useful shift in perspective.

A lot of people still talk about agents as if the main risk comes from the prompt box. The talk argues something more serious:

the real risk lives in the chain of intermediate capabilities.

Every step adds a new place where something can be:

- leaked
- manipulated
- misclassified
- falsely reconciled
- or silently accepted as plausible

---

3 Every subtask in an autonomous workflow becomes an attack surface

The bookkeeping example becomes the talk's threat-model slide.

The speaker maps each apparently normal subtask to a corresponding failure or abuse mode:

- **Reads email** → can be pushed to pull confidential emails or addresses
- **Extract invoices/receipts** → can expose customer lists
- **Extract bank statements** → can leak banking details
- **Handle duplication** → can be manipulated to mask duplicates and distort totals
- **Work out classifications** → can be used to extract broader financial statements
- **Work out reconciliation rules** → can be nudged into "fuzzy" reconciliation that misstates expenses
- **Split bills / partial payments** → can be bent into fraudulent transaction generation

This is the slide where the talk gets particularly concrete.

The argument is not just that agents may be wrong. It is that they can fail in ways that are:

- operationally harmful
- financially meaningful
- hard to notice
- and easy to disguise as normal business logic

That is a different class of risk from a normal chatbot.

A chatbot might hallucinate. A production agent may **quietly mutate a business process**.

---

4 Agents are easy to build, unsafe to deploy

The talk's main thesis appears in a slide with three risk buckets:

- **Infrastructure**
- **Authority**
- **Influence**

This is a strong framing because it separates three failure modes that people often blur together.

4.1 Infrastructure

This bucket includes the practical realities of where agents run and what they depend on:

- Macs, laptops, VPSes, or cloud boxes
- expensive model calls and background jobs
- secrets and API keys
- exposed tunnels and ports
- fragile toolchains and dependency sprawl

In other words: agent security starts before the model reasons about anything. It starts with the machine, the network, the credentials, and the process wiring.

4.2 Authority

This bucket is about what the agent is actually allowed to do:

- filesystem access
- tool installation or execution
- data access
- permission scopes
- credential access

This is a crucial distinction.

Agents are not dangerous because they sound intelligent. They are dangerous when they are given **real authority** over:

- files
- systems
- accounts
- tools
- secrets

The attack surface expands in proportion to the permissions.

4.3 Influence

This bucket is about what can steer the agent over time:

- prompt injection
- manipulated retrieved content
- conflicting tool instructions
- instruction drift across long workflows
- compounding behaviour from earlier mistakes

This is the part that most security conversations reduce everything to, but the talk treats it as only one layer of the problem.

That is correct.

An agent can be compromised not just by bad infrastructure or over-broad permissions, but by **being psychologically steerable through the very content it reads**.

---

5 The execution plane exists. The security control plane mostly does not.

The strongest line in the talk so far is essentially this:

> Open-source agents already have an **execution plane**. The **security control plane** barely exists.

That is the heart of the argument.

Today, it is relatively easy to assemble agents that can:

- call tools
- read files
- browse the web
- execute code
- move data between systems
- act on behalf of users

What is much less mature is everything needed to make those powers governable:

- capability boundaries
- policy enforcement
- reliable approval checkpoints
- transparent audit trails
- lifecycle testing
- intervention mechanisms
- shutdown and rollback paths

So the talk is not anti-agent. It is anti-naivety.

The point is:

we know how to make agents act before we know how to control them well.

That is exactly why deployment risk is rising faster than security maturity.

---

6 What a real control plane needs to do

The talk’s answer is a four-part control-plane model.

A proper control plane must be:

- **Bounded**
- **Accountable**
- **Controllable**
- **Transparent**

6.1 Bound

The system should limit access and autonomy by design.

Concrete implementation ideas from the slide include:

- isolated runtimes
- blocked commands
- allowlists
- hardened environments

This is the blast-radius discipline.

The first job of a secure agent system is not to trust the model. It is to make sure the model cannot do too much even when it is wrong, misled, or compromised.

6.2 Accountable

A control plane should preserve meaningful human oversight at important checkpoints.

That means policies for:

- confirmations
- escalation
- approval requests
- significant decision boundaries

The goal is not to force a human into every micro-step. It is to preserve human review where cost, compliance, or irreversible action matters.

6.3 Control

The talk stresses technical enforcement across the full lifecycle.

That includes:

- policy hooks
- guardrails
- ongoing tests
- mechanisms that do more than merely suggest the right behaviour

This is the difference between a guideline and an actual control.

6.4 Transparent

Finally, the system needs explainability and complete auditability.

The slide emphasizes:

- full audit trails
- logs of tool calls
- inputs and outputs preserved for investigation

Without that, incidents become impossible to diagnose cleanly.

If an agent:

- used the wrong source
- acted on injected instructions
- leaked a secret
- or performed an unsafe tool call

then the team needs to reconstruct exactly:

- what it saw
- what it decided
- what it called
- and what policy did or did not intervene

---

7 Can you actually trust your controls?

One of the most important slides so far asks a deceptively simple question:

Can you trust your controls?

The example is straightforward:

> Tell an agent: **never access confidential files**.

The speaker then asks the uncomfortable follow-up:

- does it obey **100%** of the time?
- **99%**?
- **87%**?

That is a useful correction to how teams often talk about safeguards.

People like to describe controls as if they are binary:

- the rule exists
- therefore the risk is solved

The talk argues the opposite.

With agents, a control is often better understood as a **probabilistic defence under pressure**. Its performance depends on context, attacker strategy, workload shape, and how long the workflow runs before something goes wrong.

The slide lists several variables that can change whether a control holds:

- prepend a huge amount of text before the adversarial request
- ask the agent what it can do, then craft a more targeted attack
- keep querying until the attacker finds a way to steer behaviour

That is a strong point.

A control that survives one simple prompt test may still fail when:

- the context window is crowded
- the model is fatigued by long instructions
- the attacker has learned the agent's capabilities
- or repeated attempts slowly shift the agent into a weaker state

So the right question is not:

- "Do we have a rule?"

It is:

- "What is the failure rate of that rule under adversarial conditions?"
- "How does that rate change with longer context, targeted prompts, and repeated attempts?"
- "How quickly would we notice if the control started degrading?"

This is where agent hardening starts to look more like reliability engineering than prompt writing.

The talk's framing here is especially good:

security is risk mitigation, not risk elimination.

That means a mature team should assume:

- attacks will happen
- some controls will degrade
- some defences will be bypassed
- and observability matters just as much as prevention

In other words, a control is only as real as your ability to:

- test it adversarially
- measure it continuously
- detect when it is failing
- and contain the damage when it does

---

8 Benchmark the control plane, not just the model

The talk then shifts from principle to instrumentation.

A live demo screen shows a benchmark UI titled **Determinism Benchmarks**. That is an important signal.

The point is not merely that teams should have policies. It is that they should be able to **measure whether those policies hold consistently**.

This follows directly from the previous slide.

If a control is probabilistic, then a one-off successful test means very little. A production team needs repeated evidence that the same boundary produces the same outcome across:

- repeated runs
- longer context windows
- adversarial variants
- changing model states
- and targeted attacks that adapt to what the agent can do

This is where the idea of determinism becomes operationally useful.

In an ideal world, if a policy says:

- never access confidential files
- never exfiltrate customer data
- never install unapproved tools

then the behaviour should be reliably stable under test.

If the same agent sometimes blocks a forbidden action and sometimes allows it depending on context pressure, probing strategy, or repeated retries, then the policy is not yet trustworthy enough for serious deployment.

That is why the benchmark view matters.

It turns hardening from a vague assurance exercise into something closer to reliability engineering:

- define the boundary
- run the attack scenarios
- record pass/fail outcomes
- inspect regressions
- and keep testing as the system evolves

This is a very important shift in mindset.

A lot of teams benchmark models for:

- quality
- speed
- latency
- cost

But the talk argues you also need benchmark suites for the **security control plane** itself.

That means testing things like:

- refusal consistency
- permission boundary adherence
- prompt-injection resistance under long context
- tool-call policy enforcement
- and degradation over repeated adversarial attempts

The right question becomes:

not “do we have safeguards?” but “what do the safeguard failure curves look like?”

---

9 A control plane has to be operable

The next demo screen appears to show an operator-facing interface rather than just a benchmark table.

The precise product name is hard to read from the photo, but the structure is clear:

- a chat-like control surface
- proposed security or policy text
- and a workflow for configuring or reviewing guardrails

That matters because it turns the phrase **control plane** from an architectural metaphor into an actual operational surface.

A real control plane cannot live only in:

- one prompt file
- a system message
- a policy PDF
- or a vague trust-and-safety guideline

It needs to be something operators can actually use.

That means a human should be able to:

- describe a boundary in plain language
- inspect how that boundary is implemented
- review where it applies in the lifecycle
- benchmark it against realistic adversarial tests
- and iterate when the control fails

This is a much stronger model than “write a good system prompt and hope.”

The talk’s implied argument is that a security control plane needs product-level usability.

If security teams, operators, or builders cannot:

- author policy
- review policy
- observe behaviour
- compare benchmark results
- or audit incidents

then the control plane is not mature enough to govern real agent deployments.

This is especially relevant for fast-moving teams.

Without an operable control plane, the path of least resistance becomes:

- shipping broad permissions
- relying on soft instructions

- discovering failures only after something breaks

An effective control plane should instead make it normal to:

- define controls explicitly
- test them continuously
- and revise them like any other production system component

In short:

security controls need a user interface, not just a philosophy.

---

10 Why this matters for AI-native content and media teams

This is the part that matters most for operators.

A lot of content and media teams do not think of themselves as “agent companies.” But once your workflow includes pieces like:

- research agents
- script-generation agents
- asset-routing agents
- render automations
- QA judges
- analytics feedback loops

then you already have an agent system.

That means the hardening questions apply immediately:

- Which tools can each component call?
- Which folders or systems can it read?
- Which actions require approval?
- What content is treated as untrusted?
- What logs exist after a bad output or a strange run?
- How do you tell the difference between a bad creative choice and a compromised workflow?

This is why a mature AI content stack cannot stop at generation.

It also needs:

- clear ops structure: https://instavar.com/blog/ai-production-stack/AI_Content_Ops_System_From_Brief_to_Measurement
- deterministic media pipelines where steps are inspectable: https://instavar.com/blog/ai-production-stack/Remotion_Automated_Video_Workflows
- explicit QA guardrails before publication: https://instavar.com/blog/ai-production-stack/Quality_Control_for_AI_Generated_Video_Brand_Safety_Playbook

The better the automation, the more important it becomes to govern it.

---

11 Draft takeaway so far

Based on the slides so far, the talk's central point is:

agents should be treated less like smart chat products and more like privileged workflow actors operating across infrastructure, authority, and influence surfaces.

That changes the security mindset.

The problem is no longer only:

- prompt safety
- jailbreak resistance
- output quality

It is also:

- runtime isolation
- permission design
- secret handling
- policy enforcement
- auditability
- and human override at the right checkpoints

So far, the speaker's strongest contribution is not a single clever defense.

It is the framing that:

the execution plane for agents is already real, but the security control plane is still immature.

That is the right problem statement.

---

12 Notes to expand as more slides come in

Likely sections to add next as the talk continues:

- concrete hardening patterns and implementation details
- examples of isolated runtimes / sandboxes
- approval and escalation design
- prompt injection defenses in agentic workflows
- audit log design and incident response
- tradeoffs between autonomy and usability
- production checklist / operator playbook

Last updated 11 Mar 2026. Early conference-note draft based on the opening section of Michael Hart's "Hardening Agents: Locking down the attack surface."